TECHNION
Israel Institute
of Technology

COMPOSING MUSIC BY MACHINE LEARNING

Machine Learning 2 (097209) – Project Report

Spring 2021

Abstract

In this project we will explore some ideas of working with musical sequences in ABC notation, which will be encoded to be fed as input for few models. The primary objective is to let our models process and augment a sequence based on the note until a new piece of music is produced.

In recent years, there has been an increase of interest in artificial art, Deep Fake, GANs, NLP, text generation, etc. In our project we decided to research and learn music, and in particular music generation.

At first, in Basic Part, we attempt to learn music pieces in ABC notation by using LSTM, and generate new music piece.

Then in the Advanced Part we will try to improve the Basic Part by using Transformer, instead of LSTM, in order to generate new music.

After getting the results of Basic and Advanced parts, we will choose the best one and use it in the Creative Part for another special task that interested us. In this part we want to check how our best model will deal with "inflating" existing music part.

Data

Data Import

The data that was used in this project was taken from https://github.com/IraKorshunova/folk-rnn/blob/master/data/data_v2 and contains music data in ABC notation. Each music piece is represented as three rows that contain variables and values that we will explain in next part (Data Understanding).

Data Understanding

In this data, each three rows describe a music piece, for example:

```
M:2/4
K:Cmaj
|: E > D C D | E G c 2 | G E C E | D 2 C > D | E > D C D | E G c > A | G E D C | C 2 C 2 :|
|: e > d c A | G c E 2 | G E c E | D 2 D > f | e > d c A | G E c > D | E c D c | C 2 C 2 :|
```

Figure 1

Variables explanation:

- **First row- M** is the *meter*. Information about the meter can be entered in one of three ways. In our case we use the standard form - M:6/8 or M:3/4 (for example-Figure 1).
- **Second row- K** is the key signature- Cmaj, Cmin,Cmax, Cdor, (for example-Figure 1).
- **Third row-** The "tune code" used by ABC to represent the notes. (for example-Figure 1 and Figure 2).

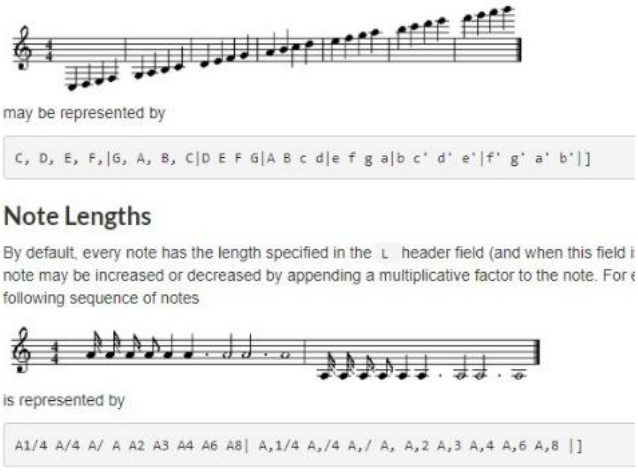

Figure 2- taken from: http://fileformats.archiveteam.org/wiki/ABC_(musical_notation)

Data Preparation

For each music piece we took the three rows- the M-variable, the K-variable and the row of the "tune code" that represents the notes (for example- n notes), and create a vector out of those three such as:

$$[M-variable, K-\ variable, tune\_char\_1, \dots\dots\dots, tune\_char\_n]$$

Basic Part

In this part we attempt to learn music pieces in ABC notation by using LSTM, and generate new music piece. To create the data set that we trained on, we divided the data to music pieces. Each music piece contains three rows: M-row, K-row, tune-code - we represent them as vector that will be the input for LSTM algorithm.

Long Short-Term Memory:

In order to learn the ABC notation music pieces we use 3-layer-LSTM. Each music piece is represented as vector of "word", where the "word" can be the M-variable or K- variable or one (or more) characters from "tune code". Given a music piece we represent it as input vector such as: $[M-variable, K-\ variable, tune\_char\_1, \dots\dots\dots, tune\_char\_n]$

Then we use the LSTM model:

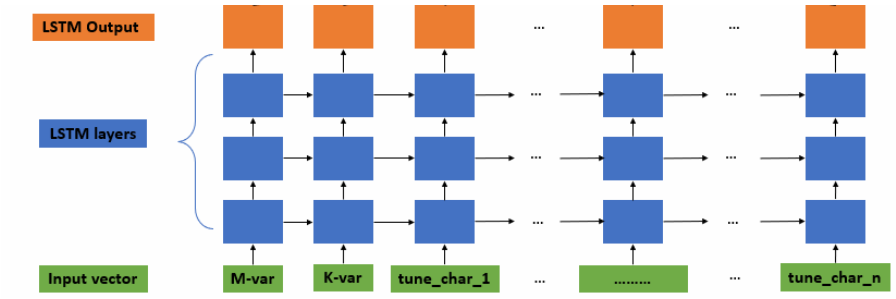

Figure 3

The output that we get is the distribution to the next "word": tune_char/ K- variable/ M- variable. By using these distributions, we predict the next "word".
In order to evaluate our model we calculate two measures:

- Loss - calculate the cross-entropy loss (Softmax) by count
- Accuracy - calculate the ratio of times in which the model predicted correctly.

Main steps of Basic Part model:

- We embedded the data "words" (tune_char/ K- variable/ M- variable) using torch Embedding.
- We combined the embedded words into vectors, each music piece into vector. These vectors are the input of LSTM. We used the LSTM and the Embedding steps as the encoder.
- Then, we applied a Linear-Layers model on the output from LSTM. This additional Linear-Layers model generates scores for next word of each given LSTM output.

The total model of Basic Part:

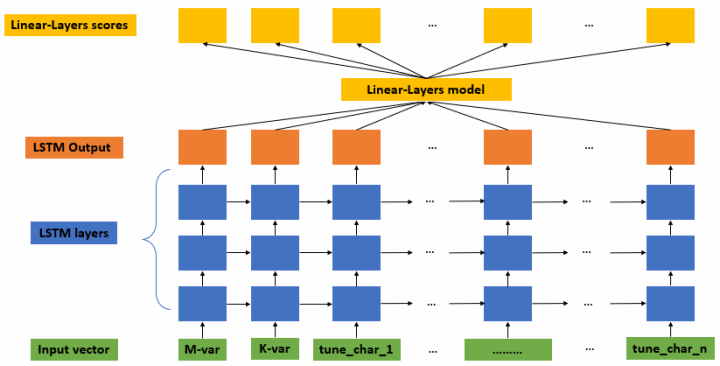

Figure 4

Parameters for total model:

After trying multiple attempts, we found that the parameters that give the best results are:

epochs	lr	lr_decay	step_lr_decay	sample_every	LSTM-hidden_size	LSTM-num_layers	LSTM-dropout

20	0.005	0.1	8	3500	262	3	0.1

Advanced Part

In the Basic Part we learned music pieces by LSTM and generated new music piece.
In this part we try to improve our basic result by using Transformer block, instead of LSTM, which gives better results according to papers (Credit in References). Based on the paper "Attention is all you need" we learned the following quote:

" The nn.Transformer module relies entirely on an attention mechanism (implemented as nn.MultiheadAttention) to draw global dependencies between input and output. The nn.Transformer module is highly modularized such that a single component (e.g., nn.TransformerEncoder) can be easily adapted/composed." (Credit in References)

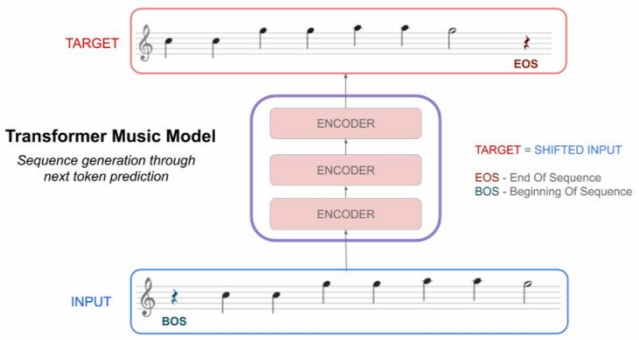

Figure 5

Main steps of Advanced Part model:

In this case, the same way as in the Basic Part, we use each music piece as a text piece and convert it to a "word vector" and use torch Embedding. Afterward we continue with the Transform model:

- We combined the embedded words into vectors, each music piece into vector. These vectors are the input of Transformer. We used the Transformer and the Embedding steps as the encoder.
- Then, we applied a Linear model as decoder on the output from Transformer. This additional Linear model generates scores for next "word" of each given Transformer output. (As in Figure 5)
- In order to evaluate our model we calculate the same measures as in Basic Part: Accuracy an Loss.

Parameters for total model:

After trying multiple attempts, we found that the parameters that give the best results are:

epochs	lr	lr_decay	step_lr_decay	d_model	nhead	Transformer - hidden_size	Transformer - num_layers	Transformer - dropout
20	8e-5	0.95	1	250	2	250	3	0.2

Creative Part

In the previous basic and advanced parts we got good results in generating new music piece. Now, in this part, we want to see how our model will deal with another new task. We want to check if we can get the same good results in "inflating" an existing music piece. The main idea is to insert new "word" between every few existing "words", so that the new "word" relies on those few previous existing words.

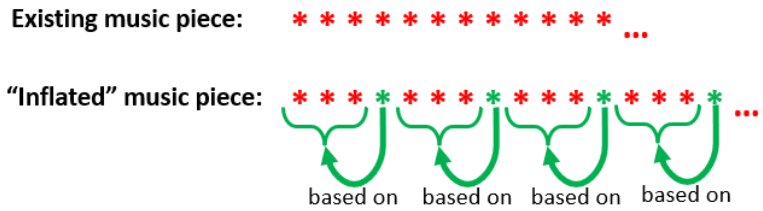

Figure 6

In other words, we want to check how our model will deal with generating and adding new "word" based on existing few "words" and not based on "words" that the model generated before. For example, every 3 "words" in the existing music piece, the model will add new "word" based on those 3 previous "words".
We decided to generate only one "word" after every few "words", so that the new one will be based only on the existing "words". This is in order to prevent a situation where every "new word" after the first one will be generated based on those that were generated before, instead of been generated based on existing "words" in the original music piece. (As in Figure 6)

Discussion and Conclusions

After all three parts we get the result of accuracy and loss evaluation matrix.

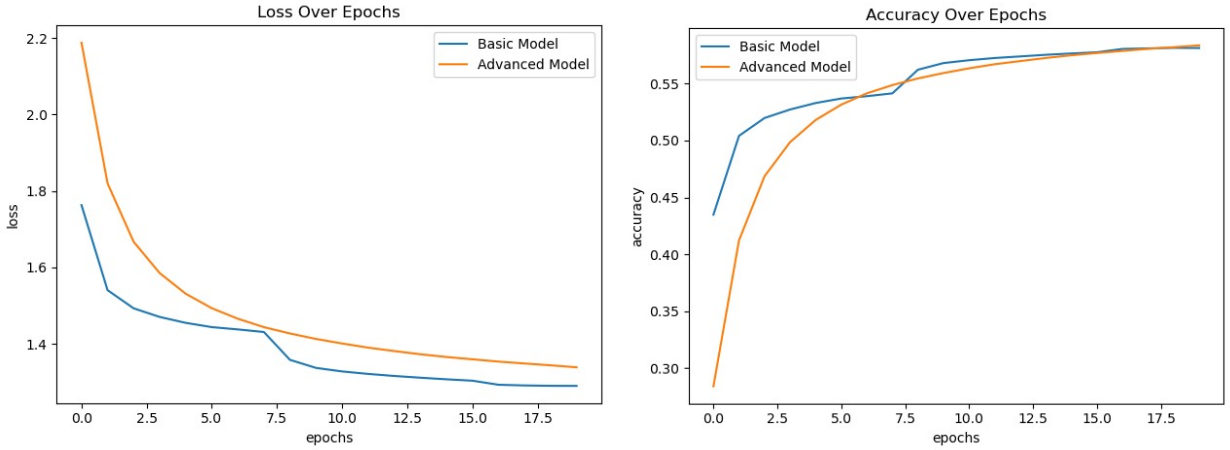

Figure 7

In addition, we can also hear our results:

Basic Part results: https://drive.google.com/drive/folders/1wnMNV8gEw9gEO35ng1_u4KTzEfhomxaD?usp=sharing

Advanced Part results: https://drive.google.com/drive/folders/1aMXh2KQ-AVlb8r8pn0e1_bfvm0ukQgCD?usp=sharing

Creative Part results: https://drive.google.com/drive/folders/1Zc-WRms-eEKo1hAsTOeCu03sD_quPWMn?usp=sharing

(The original files: "original_X.midi" , the "inflated" files: "new_X.midi")

As we can hear, the result of the Advanced part is much better than the Basic part; despite the fact that the accuracy and loss results (in the graphs in Figure 7) of the Basic part is better than the Advanced part.

There can be few reasons to those results:

- One of the reasons can be that there is an overfit in the basic part: the "basic music result" is more monotonical than the "advanced music result". That can explain the difference in the graphs, in particular the loss graph, and music pieces.
- Not enough training time or not optimize hyperparameters.
- The dataset is not big enough for such algorithms.

In the Creative Part our goal was to generate "inflated" music piece based on existed one. To do this, we used the model of the Advanced part. After few experiments we decided to skip the rows of K-variable and M-variable, and to add a new "word" after every 2 "existing words".
As a result, we got a new music piece which is based on the original one but longer and different. Because the result that we got is a new "semi-generated" music piece, we can't compare it with anything else or get any math evaluating matrix that will help us to evaluate our result.
Instead of this, we can evaluate the result by hearing the new music piece. Of course this is a subjective way, but it can still help us to estimate our results. As we can hear in our results (links in the Discussion and Conclusions) the model successfully "inflated" the original music parts and extend it, and it still sounds appropriate.

References

In our advanced part we use the next tutorial and papers as a helper material:
https://pytorch.org/tutorials/beginner/transformer_tutorial.html

https://towardsdatascience.com/creating-a-pop-music-generator-with-the-transformer-5867511b382a

https://alxmamaev.medium.com/generating-music-with-ai-or-transformers-go-brrrr-3a3ac5a04126